An Experimental Composition Tool

R.C. Mosteller
California Institute of Technology
Pasadena, California 91125

TM# 4407

February 1982

(Submitted to MICROELECTRONICS '82
A National Conference on Microelectronics
Adelaide, SOUTH AUSTRALIA, 12-14 May, 1982.)

This work supported in part by
Silicon Structures Project.

An Experimental Composition Tool

R.C. Mosteller

2 February 1982

California Institute of Technology
Pasadena, California

INTRODUCTION

With the advent of VLSI it will be possible to put 100,000 devices on a single chip. The design time of a complex VLSI chip with traditional methods is exponential at best(Moore 79, Lattin 79). The structured design methodology presents a solution.

Structured integrated circuit design is similar to the structured design of programs. The design is partitioned into small manageable pieces called cells which are similar to procedures in programming. The cell model is considered to be rectangular in shape as shown in figure 1. All geometrical data is in the interior of the rectangle and is considered to be the property of the cell. The rectangle is called a Bounding Box. Thus, the cell is self contained. The interface to the cell is through ports that are located at the perimeter of the cell. These ports protrude from the boundary of the cell like fingers.

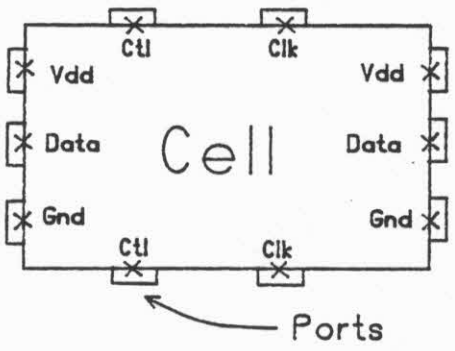


Figure 1: The Cell Model

Each cell is composed of other cells or primitive elements forming a hierarchy of cells akin to the nesting of procedures in a high level language. The number of elements in a cell is kept small, approximately seven, corresponding to the average person's short term memory(Miller 56). This hierarchy provides a method for managing a complex design.

In structured programming it is desirable to have no global variables(Wulf 73) and likewise in structured integrated circuit design global wires are undesirable. A global wire is a wire that extends through several cells and is not part of their definition. That is, such global wires are not laid on top of cells, they must be made part of the cell definition.

The cells of a structured design are partitioned into two types:

composition cells and leaf cells. A composition cell is a cell which contains other composition cells or leaf cells, while a leaf cell contains only wiring and primitive circuit elements. This partitioning of cell types is called a separated hierarchy(Rowson 80) as shown in figure 2. Note that the leaf cells are at the bottom of the tree while the composition cells form the branches. Each cell, either a composition cell or a leaf cell, contains a small number of elements that are easily grasped by a designer. Therefore, a designer can comfortably work on a single cell in the hierarchy of his chip. This separated hierarchy also provides the ability to partition the design among many designers.

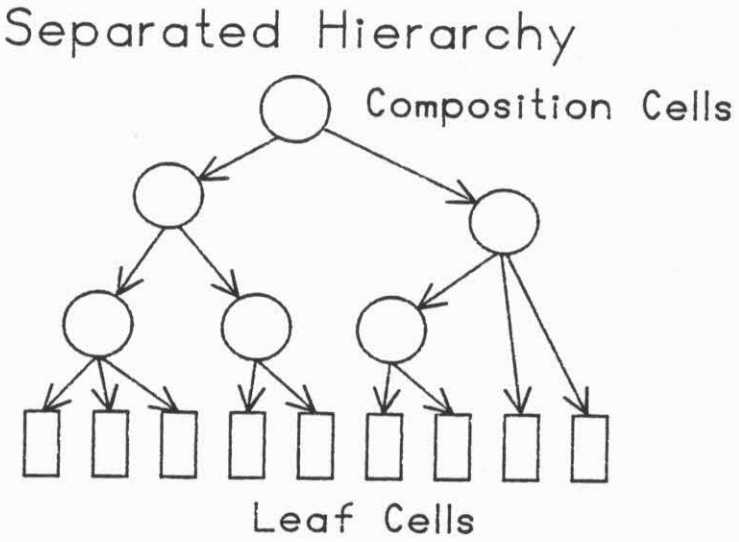


Figure 2: Separated Hierarchy

Connections to the cells occur at the perimeter of the cells at locations called ports. Cells are connected by abutment where possible. When two cells are to be joined and the ports do not line up the cells are deformed to align the ports for abutment or an intervening routing cell is added. This structured approach promotes a regular structure with a consistent wiring strategy. This design style for VLSI systems is presented in Mead and Conway(Mead 80).

There are two major divisions in a computer aided design system consistent with this design philosophy: a leaf cell design system and a composition tool for constructing the VLSI chip from leaf cells and composition cells. This computer aided design system is shown diagrammatically in figure 3. This system is a simple design system with interfaces between parts using standard text files. With this system, it is not necessary to have an exotic data base system.

The interface between the modules of the system is through a simple textual disk file using a standard form called the Sticks Standard(Trimberger 1980). This form captures the topology, layout, and structural data for cells. This standard provides a clean interface between the leaf cell design tools, the composition tool, and the verification tools. Also additional tools may be added to the system through this clean interface.

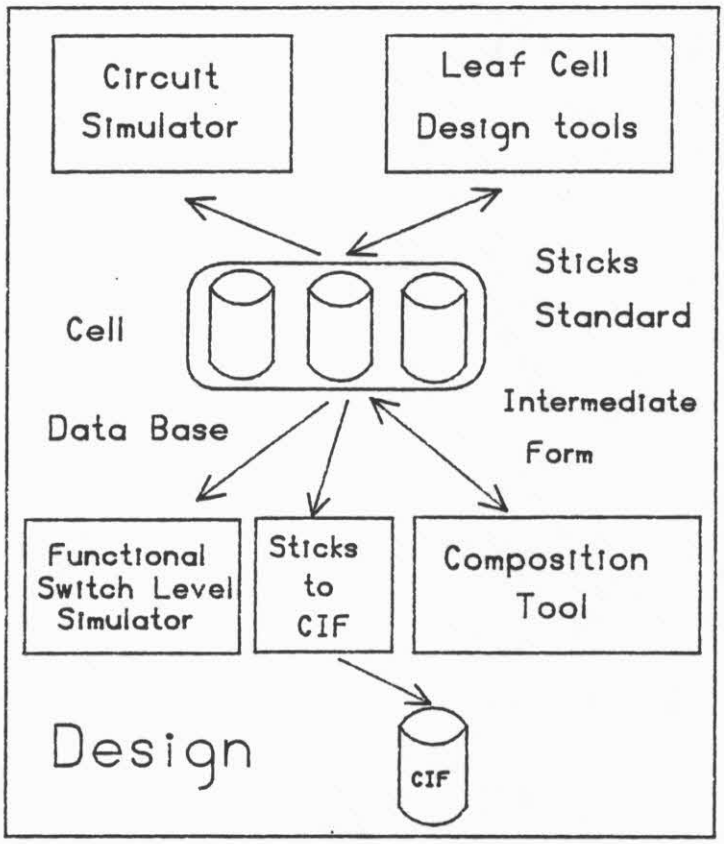


Figure 3: Design System Overview

The essential task in leaf cell design is to capture both the circuit topology and geometrical layout in the description of each cell. Leaf cells can be generated by program function such as PLA generators or by manual design. The manual creation of leaf cells is best accomplished by a sticks notation devised by Williams(Williams 77) and described in Mead and Conway(Mead 80). At Caltech there is a leaf cell design system that uses this notation called REST(Mosteller 81). The REST system provides the means to process the sticks notation, translate this notation to a abstract sticks representation, and compact the sticks respresentation. The verification of leaf cells can be accomplished by a circuit simulator for the analog domain such as SPICE or a switch level simulator(Bryant 81) for the logic domain. Since the leaf cells are constructed via program function or by a sticks system they may be guaranteed free of design rule errors.

The task of assembling the chip is performed by a composition tool. The input to the composition tool is a set of leaf cells and composition cells. Composition instructions define the spatial positioning and structure layout of the design. The description of the composition is best accomplished in a textual manner with a composition language. The main point here is that the spatial positioning and structural data is captured in a simple manner at one place in the design description.

The actual composition or tiling of the VLSI chip is performed by aligning the pitches of the cells or by adding wiring cells. Alignment is performed by stretching a cell to to fit its environment. The stretching is performed by the composition tool since detailed knowledge of the geometry is not necessary in the stretching process. Only the wiring and primitive elements are required. Compaction could also be used in the alignment process by adding constraints to the leaf cells and using the REST system to process the cell.

The verification is accomplished with a switch level simulator(Bryant 81) for the logic domain or the circuit simulator SPICE may be used for the analog domain. The Sticks Standard is automatically translated to the proper input format for the verification tool.

When the design is ready to be fabricated the Sticks Standard file is translated by a program to the design description in text format known as the Caltech Intermediate Form (CIF)(Mead 1980). CIF can be processed by various silicon foundry tools to produce mask geometry.

COMPOSITION CELL DESIGN

The composition of cells in a design is governed by a set of composition rules(Rowson 80). The use of these rules will insure a design rules error free chip. Two cells can be connected by abutment along one edge providing there is a one-to-one correspondence of ports both in number and in layer. Both A and B in figure 4 meet this requirement. Notice in A that although the ports do not correspond in location they do correspond in sequence. In B the ports are colinear so that abutment will be sufficient for composition. In A there are two choices for composing the cells: 1) the cells may be stretched so that the ports are colinear and the cells abutted or 2) a river routing cell may be added between the two cells. A combination of these two strategies may be also be employed. The result of the composition is a new cell with the same properties as the cell model in figure 1.

The bounding rectangle for the cell defines the edges for its juxtaposition with other cells. The bounding box for leaf cells and composition cells is extended one half the maximum design rule from all interior elements. In future systems the cells will have a bounding perimeter for each logical layer.

To compose two cells, the composition system only requires the names of the two cells to be composed and the edge for abutment. Coordinates are not needed. This type of abutment composition is best described in a textual manner. Abutment composition requires two binary operators: one for vertical composition and one for horizontal.

The experimental composition tool, **RCOMP** is embedded in the SIMULA programming language(Birtwistle 73). SIMULA is a block structured language similar to ALGOL with the addition of classes. A class is an object that includes both data and code. An embedded language approach for composition was used for programming expediency and for ease of change of the system. Also, various composition constructs could be experimented with. The drawback of an embedded language is that a user must compile and execute a program for each composition. In addition, the composition constructs must conform to the syntax of the embedded language where a composition program may have a clearer syntax for the

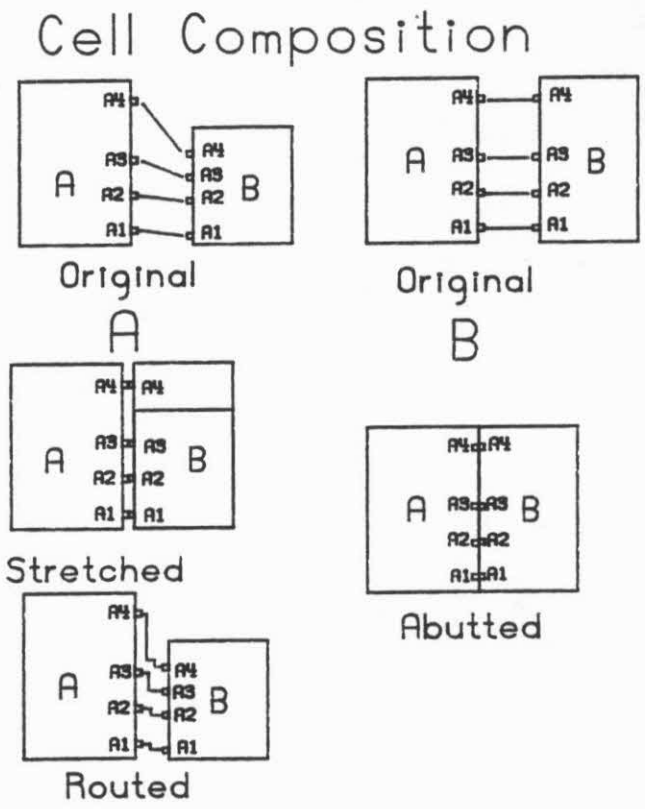


Figure 4: Cell Composition

users.

The following will describe the statements and their use in the RCOMP system. The definition of a composition cell is accomplished with the define and enddef statements. The keyword DEFINE(<cell name>) opens the definition of a cell while the keyword ENDDEF closes the definition of cell. The cell name is the name of the cell in the process of being described. The cell names must be unique and RCOMP checks for cell name uniqueness. The definitions may be nested. Any closed composition cell definition may be used in a subsequent composition definition. The definition of the cell must be complete prior to the close. A complete definition is one in which all the instances in the composition are joined. RCOMP checks for proper completion of cell composition.

The CALL(<cell name>) statement creates a specific instance of a cell. An instance is a reference to the cell definition, a template of the cells ports and a bounding rectangle. The cell name is the name of the cell to be instanced. Two instances are composed with the binary operators: JOIN for horizontal composition or JOINTOP for vertical composition. These operators are placed between call statements or in composition expressions separated by dots. In the top half of figure 5 cell A is horizontally joined with cell B. The result is a composite instance that has the same properties as a single instance, that is, a template and a bounding rectangle plus a list of instances that were used to construct the composite instance. In the lower half of figure 5 cell A is joined horizontally to cell B with the result being joined

vertically to C to produce a composite instance as shown. A single composite instance is required in a cell definition prior to the cell close with the ENDDEF. The precedence of the join operators is from left to right. This order may be controlled by the use of parentheses.

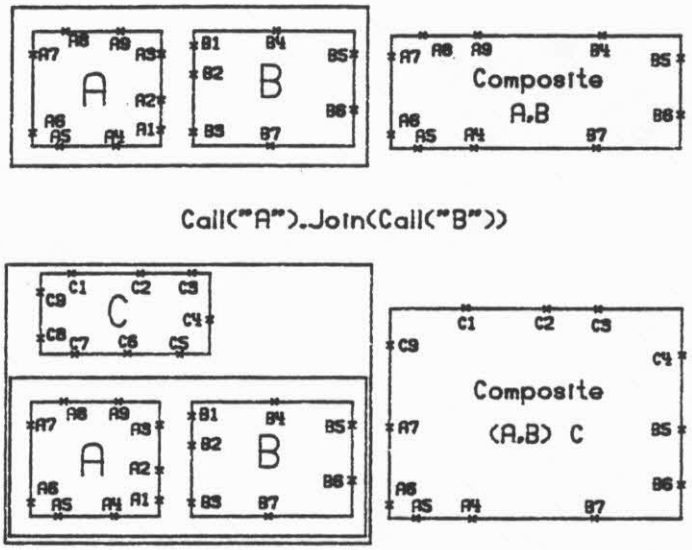


Figure 5: Composition Operators

RCOMP performs the join operation by first checking for compatibility between the sets of connectors on the edges of the instances being composed. For horizontal composition, the left instance's right connectors are checked against the right instance's left connectors. For vertical composition, the bottom instance's top connectors are checked against the top instance's bottom connectors. An instance may be a call or a composite instance created by a previously executed join operation. Compatibility between edges requires the same port count and corresponding port on each edge must have the same layer. If the edges are not compatible RCOMP will give an error message.

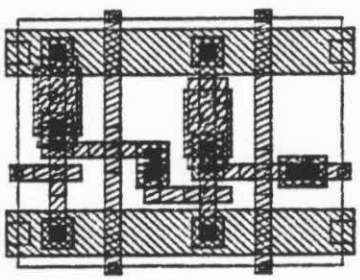
It may be necessary to mirror or rotate an instance. RCOMP supports three unary operators for this purpose: **MIRRORX** for mirroring on the x-axis, **MIRRORY** for mirroring along the y-axis, **ROTATE** for quadrant rotation. These unary operators are applied to an instance or a composite instance. Therefore, two cells may be first composed and then second rotated or mirrored and then finally joined to another structure.

In many designs, cells are replicated in rows or columns. An example would be a block of shift register cells. The boundry cells of this type of structure generally require some special connector handling. For instance, a general shift register cell with control ports on the top and bottom would require the omissions of the control ports on the boundary cells for the top of a block of shift register cells if the control ports were used at the bottom of the structure. RCOMP supports a port omit operator to allow deletion of this type of port. The **OMIT** takes as its arguments either a port number or a port name along with a

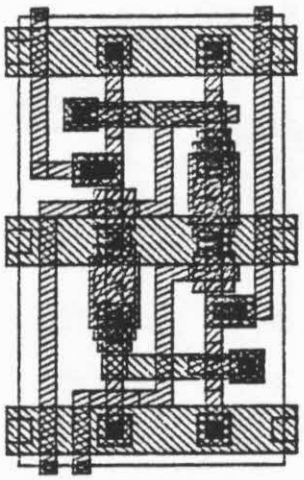
side. The omit can be used with either an instance or a composite instance.

The SIMULA looping constructs can be used for the construction of rows, columns, and blocks of cells. RCOMP supports one type of pointer that can reference an instance or composite instance. This instance pointer is used in repetitive constructs or where separate structures are constructed and merged within a cell definition.

The example in figure 6 shows two cells, a shift register cell and a super buffer in NMOS technology. The rectangular outline around each cell is the bounding box. First, a column cell is constructed with a super buffer and two shift register cells on top. The omit operator is used to delete the unnecessary control ports. An instance pointer P is also used. Second, a shift register cell is constructed by joining two columns cells just previously defined.



Cell S



Cell SB

```
DEFINE("col");
    p:=Call("sb");
    p:=p.Jointop(Call("s")).Jointop(Call("s"));
    p.omit(topside,1).omit(topside,2);
ENDDEF;
DEFINE("shiftreg");
    Call("col").Join(Call("col"));
ENDDEF;
```

Figure 6: Shift Register Cell Example

The result of the composition is shown in figure 7. Notice that the control lines on the top of the cell were peeled back to the pass transistor. The cell may now be used as a building block for other structures. This is a simple example used for illustrative purposes only.

A leaf cell or composition cell can be simply stretched if it contains only orthogonal geometry. The simplest stretch operation uses

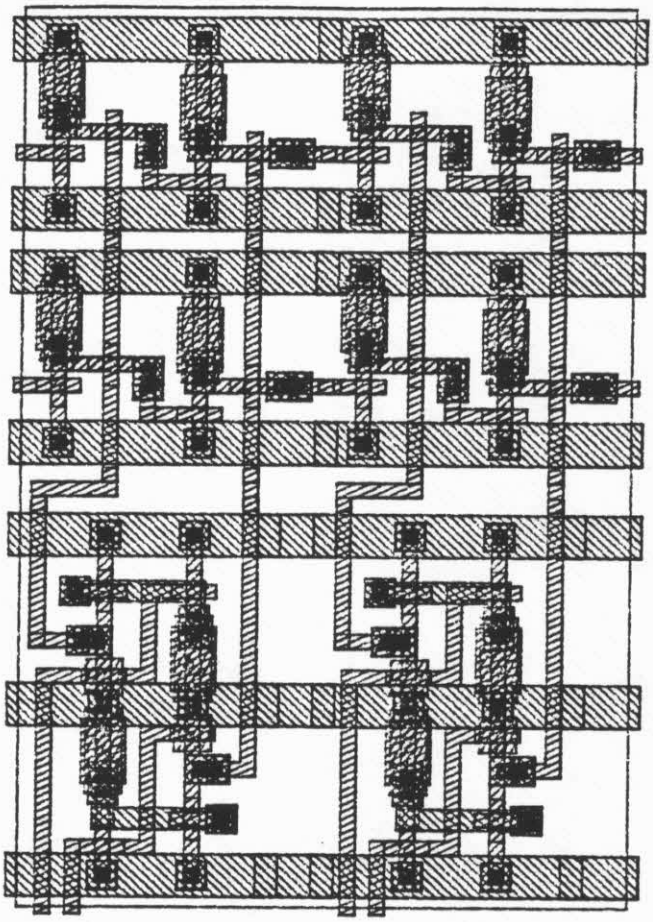


Figure 7: Shift Register Cell Example Block

rift lines as cuts through the cell. The rift lines are determined by the port boundaries. RCOMP checks each cell during the initialization process marking non-stretchable cells. Other cells may be marked by the user as non-stretchable based on circuit rules or user preference.

The join operators will be described in general terms since the **JOIN** and **JOINTOP** are symmetrical. Instances are processed immediately upon execution of the join operator. If the connectors of two instances line up, the cells are abutted otherwise the following algorithm is used. Join proceeds in two phases: first adjustment of cell pitch, and second either routing or stretching based on an area cost function. To adjust the pitch the smaller of the two cells is stretched or aligned to approximate the pitch of the larger. Pitch adjustment is effected first since the area constructed from the pitch mismatch is of zero cost. This zero cost area can be seen as the area above B to the top of A in figure 8 as the original. If the instances to be composed are of equal length along the joining edge adjustment is not required.

The cell pitch adjustment is shown in figure 8. The original shows the two cells A and B to be horizontally composed. In the stretched case, B is stretched to align to A without exceeding A's height. The innner box in B is its original size. B is first translated so that the

first port lines-up with the first port in A, then stretched. In the align case, B is non-stretchable. In this case, B is moved to align to A for minimum routing area without exceeding A's height.

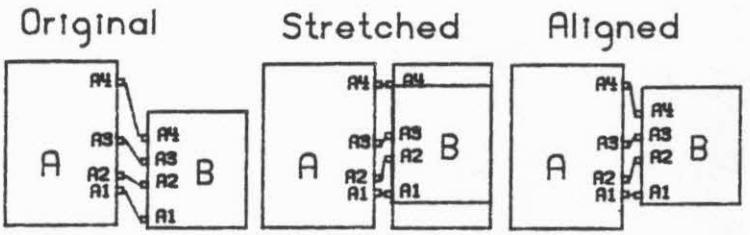


Figure 8: Cell Pitch Adjustment

The second phase first determines a measure of area cost for both routing and stretching. Based on the minimum cost, routing or stretching is performed. For routing this area would be the routing cells area. For stretching this area would be the additional area used by the cells in order for abutment to occur. The determination of stretching cost is found from the stretch tables. Two stretch tables are constructed, one for each instance or composite instance. The tables contain one entry for each port. The entry defines the amount of deformation for each port in order to abut. These tables are based on Rowson's(Rowson 80) recursive placement technique. The determination of the routing costs is defined from a pseudo route. If either cell is non-stretchable then the cells are routed.

If an instance is to be stretched then its instance template and bounding rectangle is stretched. The actual stretching and duplication of the cell definition is delayed until the wrap up phase of RCOMP. Delayed stretching is performed so that RCOMP will do a minimum amount of work and minimum cell duplication. In the wrap up phase of RCOMP, the instances of each kind of cell will be collated, merge, and then stretched.

The stretching of composite instances is accomplished in a manner similar to that of the instance since a composite instance only contains a port template, bounding rectangle, contained instances and routing data. The composite port template and bounding box is deformed as in the case of the simple instance. Contained instances are translated for all rift lines defined by the stretch table that occur below and including the lowest port on the axis of stretching. For rift lines that are in the bounds of the instance normal stretching is accomplished as in the case of the simple instance. All rift lines above the instance are ignored. This method insures that the contained instances are only stretched where necessary.

The river routing algorithm used is a modified orthogonal Tompa method(Tompa 80). This method gives the optimum solution possible with orthogonal wires. The ports define the wire widths and layers used during the routing process. The router uses the minimum of the two port widths.

The final stretching of the cells is accomplished during the wrap up phase of RCOMP. The wrap up phase collates all instances of the same cell definition with equivalent stretched templates. The original cell

is then duplicated, and deformed for each unique stretching. This method insures a minimum of cell duplication.

RCOMP was implemented on a DEC system 2060 using the SIMULA programming language. RCOMP is currently used for NMOS with Mead and Conway design rules. RCOMP, REST and the various other tools are table driven so that other technologies may be incorporated. The Sticks Standard has provided the means to modularize the system and share various design tools. There is a suite of programs that interface the Sticks Standard in addition to the ones already mentioned(Trimberger 81).

DESIGN UPDATE

The design process begins with initial design phase followed by a period of design refinement. Any design aid should be amenable to this update process. There are two types of updates to the system described: leaf cell updates and composition cell updates. The leaf cell changes involve either updating the leaf cell with the sticks editor or using the function leaf cell generator. Then the composition program needs only to be re-executed providing the tiling has not changed. The number of ports may change in the leaf cells provided the changes are consistent among all cells which connect to it.

The alteration of the composition requires editing the composition statements within the program. Any text editor would suffice for this task. Then a compilation and execution must be performed to complete the update process.

CONCLUSION

RCOMP was created to explore alternatives for cell composition. The system has been used to do a experimental design at Caltech in the Silicon Structures Project of the INTEL 8251A chip. A portion of the chip is shown in figure 9. This portion has not been optimized however it does give a view of capabilities of RCOMP. This design required about 3 pages of composition code. Various alternate tiling structure were explored with ease because of robustness and conciseness of RCOMP. The compactness and few commands has made RCOMP a joy for the users.

Future work will be in explorations of compaction algorithms coupled with composition.

REFERENCES

[Birtwistle 73]
Birtwistle G.M., Dahl O-J., Myhrhaug B. & Nygaard K.
SIMULA begin.
Auerbach Publishers Inc.,, 1973.

[Bryant 81]
Randal E. Bryant.
A Switch-Level Simulation Model for Integrated Logic Circuits.
PhD thesis, Massachusetts Institute of Technology, 1981.

[Lattin 79]
Bill Lattin.
VLSI Design Methodology: The Problem of the 80's for Microprocessor Design.
In Charles L. Seitz, editor, Very Large Scale Integration. California Institute of Technology, Pasadena, California, 22-24 January 1979.

[Mead 80]
C.A. Mead and L.A. Conway.
Introduction to VLSI Systems.
Addison Wesley, 1980.

[Miller 56]
G.A. Miller.
The magical number seven, plus or minus two: some limits on our capacity for processing information.
Psychology Review :81-97, 1956.

[Moore 79]
Gordon E. Moore.
Are We Really Ready for VLSI?.
In Charles L. Seitz, editor, Very Large Scale Integration. California Institute of Technology, Pasadena, California, 22-24 January 1979.

[Mosteller 81]
R.C. Mosteller.
REST A leaf Cell Design System.
In John P. Gray, editor, Very Large Scale Integration. University of Edinburgh, Edinburgh, Scotland, 18-21 August 1981.
ISBN 0-12-296860-3.

[Rowson 80]
J.A. Rowson.
Understanding Hierarchical Design.
PhD thesis, California Institute of Technology, 1980.

[Tompa 80]
M. Tompa.
An Optimal Solution to a Wire-Routing Problem.
In Prceedings of the Twelfth annual ACM Symposium on Theory of Computing. ACM, 28-30 April 1980.

[Trimberger 81]
S. Trimberger, J. Rowson, C Lang, J. P. Gray.
A Structured Design Methodology and Associated Software Tools.
IEE Transactions on Circuits and Systems CAS-28(7), July, 1981.

[Williams 77]
John Williams.
Sticks - A New Approach to LSI Design.
Master's thesis, Dept. of Electrical Engineering and Computer Science M.I.T., June, 1977.

[Wulf 73]
W. Wulf and M. Shaw.
Global Variables Considered Harmful.
SIGPLAN Notices , February, 1973.

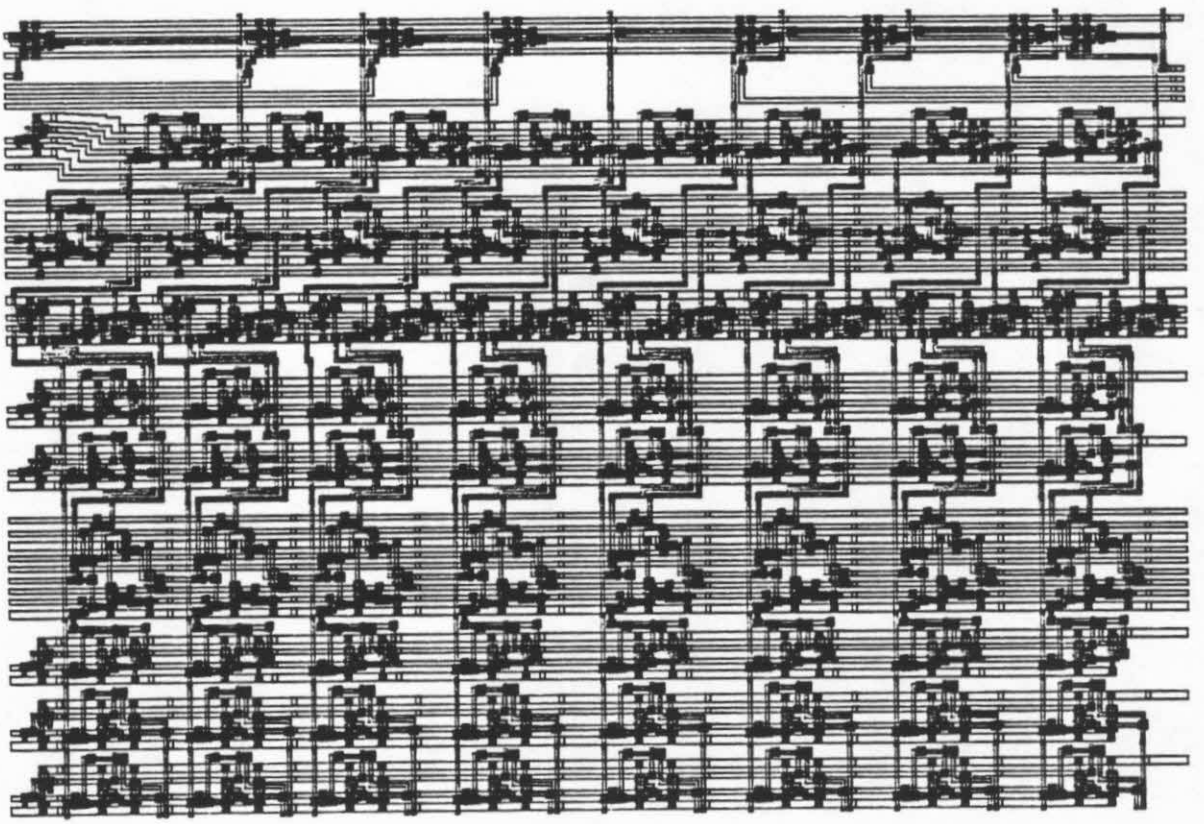


Figure 9: Example of Cell Composition